5. Anticipating Change

The preceding chapters have described an expansive methodology for building software-based products based on the concept of a product family. This methodology targets the producibility vision for a fundamental change from the traditional concept of the tool-supported building of unique, hand-crafted products. This chapter considers how potential technological advances could increase the effective capabilities of this methodology. These potential advances target intrinsic weaknesses of existing practices, methods, and technology as well as possibilities for more fundamental innovations leading toward producibility.

The increasing relevance and importance of software capabilities to all aspects of modern life have led to enormous advances in the complexity of software-based products that can now be built. In turn, this dependence on software has increased the urgency of making software more capable, more reliable, more sustainable, and less costly. Many opportunities exist for achieving responsive improvements both in how software is built and in the breadth and complexity of its effective capabilities. Such improvements depend on achieving feasible advances in many aspects of how software is built.

Advances that would enhance the practice of software-based product development, improving both process quality (productivity and predictability) and product quality, are considered in four categories:

- Technical challenges in developmental and operational practices
- Predictive analytics for more predictable product properties
- Evolving and emerging computational technologies
- Data science, artificial intelligence, and machine learning capabilities

In the beginning, software development was focused on building products having precisely defined functionality. As experience increased, software functionality became more complex and responsive to how broader practices. In time, the need to evolve software as needs and technologies changed gained in importance. At the same time, the benefit of similar enterprises using common products and compatible data formats became clear. Still different tradeoffs in product behavior may be most beneficial to different, possibly competing enterprises. Enterprises, while having similar needs, may still be best satisfied by similar rather than the same products. In either case, where a common product or multiple similar products, best meet the needs of customers in a coherent market, there remain many challenges with both of these perspectives that emerging technologies foreshadow improvements in how products are built. Here we consider technological advances that are needed and/or likely to occur in the near- to medium-term that may enable or even dictate changes in how software-based products will be built.

Software development as a science, engineering, and manufacturing discipline is continuing to change and mature, having been widely applied only since the 1950s. Theory both influences practice and provides a theoretic basis for observed advances in practice. Still, actual practice arguably exceeds in some aspects and fails in others to fully achieve the various theoretical possibilities. Current practice is most effective with small competently qualified development staff when not overly constrained by imposition of arbitrary standards of practice.

Limits on the Future Relevance of the Described Methodology

{example of device-based platforms populated by selection of specialized apps, resulting in a customized product} {computational platform can be configured as a cohort of existing components} {platform defines common standards for apps to meet}

Is future approach a scale-up of the Unix model of connecting powerful highly specialized apps (or independent purpose-built apps or components), not requiring any complex “programming” but just scripting of control and data transfers? improvements come mainly from improving (or replacing) the apps or components that have many users. is there still value in model-based (i.e., domain-specific) development? (eg apps:

Photoshop, Revit, Word/Excel, Pages/Keynote, databases, games/underlying game engines) (still a need for a level that does specialized integration of complex apps? eg a "project" app that correlates assets of different roles)

Detriments that platform must avoid: lack of a consistent user experience (different products have differing interface concept); non-congruent data definitions, resulting in fragmented, duplicate, but differing information content according to each product's scope of interest; duplicate/differing/conflicting implementations of platform capabilities